LVDC
Tri Core Architecture

- All instructions have a 32-bit format

Tasks and Contexts

- A task is an independent thread of control. There are two types: Software Managed Tasks (SMTs) and Interrupt Service Routines (ISRs).
- SMTs are created through the services of a real-time kernel or Operating System, and are dispatched under the control of scheduling software.
- ISRs are dispatched by hardware in response to an interrupt. An ISR is the code that is invoked directly by the processor on receipt of an interrupt

Each task is allocated its own mode, Determined by PSW register

- User-0 Mode: Used for tasks that do not access peripheral devices.
- User-1 Mode: Used for tasks that access common, unprotected peripherals
- Supervisor Mode: Permits read/write access to system registers and all peripheral devices. (Default)

General Purpose and System Registers

- There are two types of Core Register, the General Purpose Registers (GPRs) and the Core Special Function Registers (CSFRs).
- GPRs consist of 16 general purpose data and 16 general purpose address registers.
- The CSFRs control the operation of the core and provide status information about the core.

ENDINIT Protection

- Some CSFRs can only be modified in the initialization state, these are described as ENDINIT protected

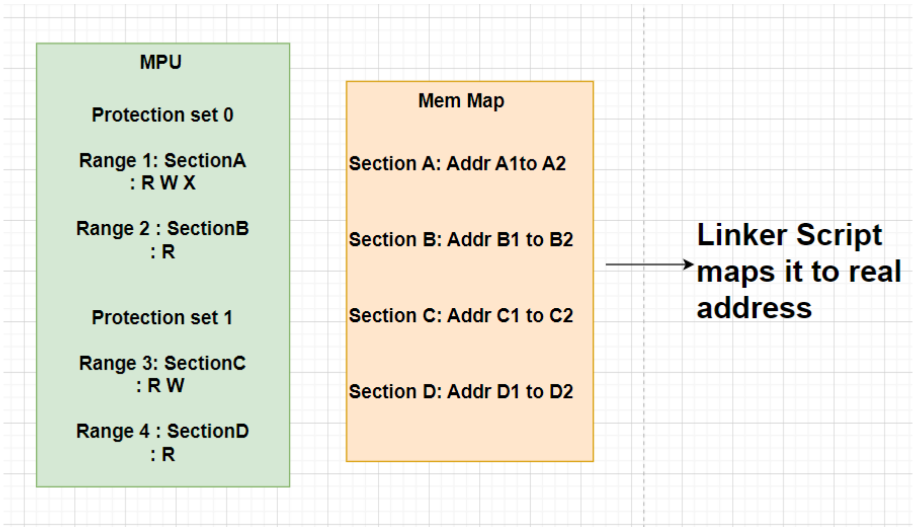
General Purpose Registers (GPRs)

- 16 Data registers (D[0] to D[15]) , 16 Address registers (AGPRs), (A[0] to A[15])
- Two consecutive even-odd data/address registers can be concatenated to form eight extended-size registers in order to support 64-bit values (E[0],E[2]..., AND P[0],P[2]..).
- A[0], A[1], A[8], and A[9] are defined as system global registers. Their contents are not saved or restored across calls, traps or interrupts

- A[10] is used as the Stack Pointer (SP).
- Register A[11] is used to store the Return Address (RA) for calls and linked jumps, and to store the return Program Counter (PC) value for interrupts and traps

Program State Information Registers

- Program Counter (PC): holds the address of the instruction that is currently running. The PC should only be written when the core is halted.
- Program Status Word Register (PSW):
    - Protection Register Set (0-3): We use only 0 and 1. One protection set will contain certain ranges of data and It will be made active when that memory is accessed



    - Access Privilege Level Control (I/O Privilege): user0,user1 and supervisor mode
    - Interrupt Stack Control:

      0- User stack, the stack pointer register is loaded from the ISP register before execution starts at the first instruction of the ISR.

      1- Shared Global stack, the current value of the stack pointer is used by the Interrupt Service Routine (ISR).
- Previous Context Information and Pointer Register (PCXI): Previous Context Information Register (PCXI) contains linkage information to the previous execution context, supporting interrupts and automatic context switching. The Previous Context Pointer (PCX) holds the address of the CSA of the previous task.

Context Save Areas

- The state of a task is defined by its context. Contexts, when saved to memory, occupy 16 word blocks of storage, known as Context Save Areas (CSAs).

- Context switching occurs when an event or instruction causes a break in program execution. The processor uses that task's context to re-enable the continued execution of the task.
- Eg: When an interrupt comes value of certain registers like PC are copied to CSA. After the ISR is executed these registers copy the original value from CSA before returning to main program
- The architecture uses linked lists of fixed-size Context Save Areas. Each CSA can hold exactly one upper or one lower context. CSAs are linked together through a Link Word
- Unused CSAs are linked together in the Free Context List (FCX). CSAs that contain saved upper or lower contexts are linked together in the Previous Context List (PCX).

Interrupt System

- The interrupt system is built around programmable Service Request Nodes (SRNs).
- A service request may come from an on-chip peripheral, external hardware, or software.
- Service requests are prioritized, and prioritization allows for nested interrupts
  Rules:
  - A service request can interrupt the servicing of a lower priority interrupt
  - Interrupt sources with the same priority cannot interrupt each other
  - The Interrupt Control Unit (ICU) determines which source will win arbitration based on the priority number All Service Requests are assigned Priority Numbers (SRPNs). Every ISR has its own priority number.
- Each interrupt source is assigned a unique interrupt priority number known as the Service Request Priority Number (SRPN).
- SRPN is used by the Interrupt Control Unit (ICU) to prioritise between multiple concurrent interrupt requests.
- SRPN of the winning request is supplied to the CPU as a Pending Interrupt Priority Number (PIPN) along with an request trigger.
- CPU checks the state of the global interrupt enable bit ICR.IE, and compares the current CPU priority number ICR.CCPN against the PIPN. The CPU can be interrupted only if ICR.IE == 1 and PIPN is greater than CCPN.
- If accepted, CPU responds with an Interrupt Acknowledge and the returns the priority number of the taken interrupt. The ICU will then clear down the requesting interrupt
- ICU Interrupt Control Register (ICR): It holds,
  - Current CPU Priority Number (CCPN)
  - The global Interrupt enable/disable bit (IE)
  - Pending Interrupt Priority Number (PIPN)

- Interrupt Vector Table:
    - The Interrupt Vector Table is an array of Interrupt Service Routine (ISR) entry points.
    - When the CPU takes an interrupt, it calculates an address in the Interrupt Vector Table that corresponds with the priority of the interrupt (the ICR.PIPN bit field). This address is loaded in the program counter. The CPU begins executing instructions at this address in the Interrupt Vector Table.
    - Depending on the code size of the ISR, the Interrupt Vector Table may only store the initial portion of the ISR, such as a jump instruction that vectors the CPU to the rest of the ISR elsewhere in memory.
    - Base of Interrupt Vector Table register (BIV) stores the base address of the Interrupt Vector Table
    - Interrupt vectors are ordered in the table by increasing priority

Trap System

- A trap occurs as a result of an event such as a Non-Maskable Interrupt (NMI), an instruction exception or illegal access.
- The TriCore architecture contains eight trap classes
- Each trap is assigned a Trap Identification Number (TIN) that identifies the cause of the trap within its class.
- Specific traps are distinguished by a Trap Identification Number (TIN) that is loaded by hardware into register D[15] before the first instruction of the trap handler is executed
- The BTV register specifies the Base address of the Trap Vector table
- The Trap Class Number (TCN) used to index into the trap vector table

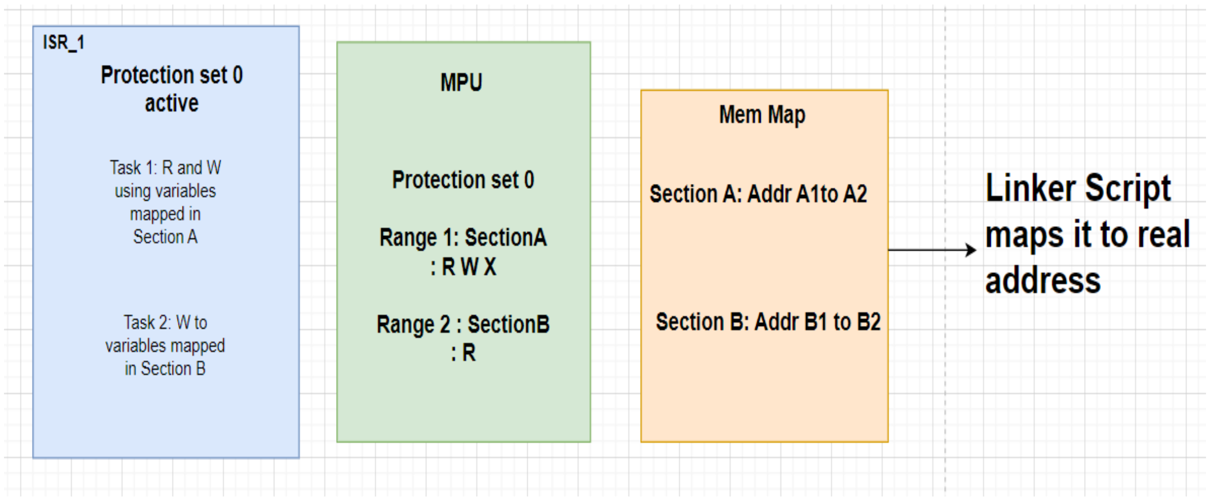
Memory Protection System

- The Trap System
- The I/O Privilege Level (user0, user1, supervisor)
- The Memory Protection System
    - Range Based
    - Page Based

Range Based Memory protection System

- A Protection Range is a continuous part of address space for which access permissions may be specified.
- A Protection Range is defined by the Lower Boundary and the Upper Boundary.

- An address belongs to the range if: Lower Boundary <= Address < Upper Boundary
- There are two groups of Protection Ranges:
    - Data Protection Ranges specify data access permissions
    - Code Protection Ranges specify instruction fetch permissions
- Access Permissions: Access Permissions define the kind of access allowed to a protection range (Read/Write/Instruction Fetch)
- Protection Sets (0-3): A complete set of access permissions defined for the whole address space used, is called a Protection Set.
- When two ranges intersect, the intersecting regions will have the permission of the most permissive range.
- When protection system is enabled (SYSCON.PROTEN == 1)., every memory access (read, write or execute) is checked for legality before the access is performed
    - Protection Enable bit in the SYSCON register (SYSCON.PROTEN)
    - Currently selected protection register set (PSW.PRS)
    - Ranges selected in the protection register set
    - Access permissions set for the ranges selected for the protection set
- There are three traps generated by the range based memory protection system, each corresponding to the three protection mode register bits:
    - MPW (Memory Protection Write) trap = WE bit
    - MPR (Memory Protection Read) trap = RE bit
    - MPX (Memory Protection Execute) trap = XE bit


ISR_1
Protection set 0 active
Task 1: R and W using variables mapped in Section A
Task 2: W to variables mapped in Section B
MPU
Protection set 0
Range 1: SectionA : R W X
Range 2 : SectionB : R
Mem Map
Section A: Addr A1to A2
Section B: Addr B1 to B2
Linker Script maps it to real address

SVN Details

- Application layer components are generated using MATLAB, Simulink
- The generated C code is placed in 40_Appl, The Simulink models are available in 10_Config ->Simulink

- 20_Make contains files related to building and flashing
- 10_Configuration -> MCAL_Configuration contains MCAL modules related to startup tests. It only contains safety related files and is generated through tresos
- In 30_BSW, The SMU,TstM and Tsthandler are used for this safety tests for testing whether the uC and firmware is working correctly or not. The code for these modules are provided by Infineon (static). Some changes are made it is configured using tresos (config). For Eg: Infineon provides lots of safety tests, we only use some of them which is configured using tresos.
- 70_Tools
    - 00_Batch: To create S drive
    - 10_Ar Tool Chain: Files from Infineon. TC21x- library for Infineon registers
    - 20_Tresos: For configuring BSW layers. (SMU, TstM , Tsthandler)
    - 31_UDE: For flashing and debugging
    - 32_Lauterbach: Not used by developers, mostly used for testing
    - 42_DeltaMakeSupport: For 20_Make

System Start Up – SSW

- Startup Software is the first software executed after a chip reset
- SSW start address in BootROM is the reset value in Program Counter register
- The last SSW instruction performs a jump to the first user code instruction
- SSW is provided by Infineon in file **IfxCpu_Cstart0.c** (We make minor changes)
- The Startup Software contains procedures to initialize the device
- SSW execution can be triggered by different events. SSW recognizes the triggering (reset) event and takes (partially) different execution flows
    - Power-on
    - System reset
    - Application reset

SSW Flow

- Test Stack, CSA and Cache – Not Used
- Clearing all DSPR(Data Scratch Pad RAM)

- Loading User stack Pointer
```
__setareg(sp, __USTACK(0));
__dsync();
```

- PSW(Program Status Word) is set to Default Value
```
__mtcr(CPU_PSW, IFXCSTART0_PSW_DEFAULT);
```

*/*mtcr: Move to core register. For moving contents of a data register to the addressed Core Special Function Register. CPU_PSW is the program status word register and it is set to default value*/*

- Disabling Program Cache , Data Cache

  */*enable/disable program cache depending on the configuration */*
  ```
  IfxCpu_setProgramCache(FALSE);
  ```
  */*enable/disable data cache depending on the configuration */*
  ```
  IfxCpu_setDataCache(FALSE);
  ```

- Initialize the clock system
- Loading BTV – Trap Vector Table

  / Load Base Address of Trap Vector Table. */*
  ```
  __mtcr(CPU_BTV, (uint32)__TRAPTAB(0));
  ```

- Loading BIV – Interrupt Vector Table

  / Load Base Address of Interrupt Vector Table. we will do this later in the program */*
  ```
  __mtcr(CPU_BIV, (uint32)__INTTAB(0));
  ```

- Loading ISP

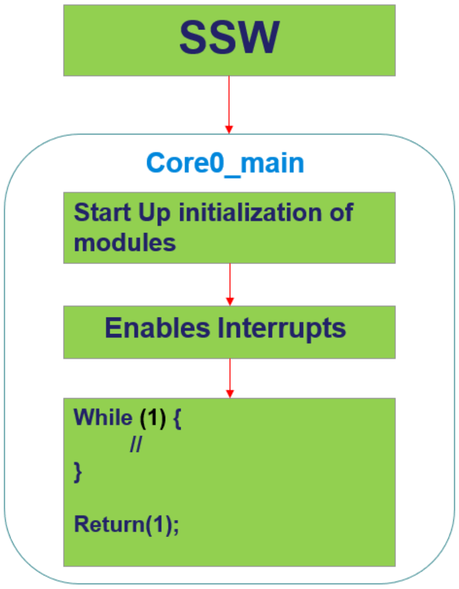
  / Load interupt stack pointer. */*
  ```
  __mtcr(CPU_ISP, (uint32)__ISTACK(0));
  ```

- Disable CPU and Safety WDG (internal WD) at the start-up as they are not used
  ```
  IfxScuWdt_disableCpuWatchdog(cpuWdtPassword);
  IfxScuWdt_disableSafetyWatchdog(safetyWdtPassword);
  ```
- Jump to main program
  ```
  __non_return_call(core0_main);
  ```

Note:

Infineon microcontroller stack has 2 sections:
USTACK: User Stack: Store local variables when calling sub function in a function. The local variables are pushed into USTACK when entering the sub function and popped out when exiting the sub function.
ISTACK: Interrupt stack: Store local variables before the code is interrupted by an ISR. The local variables are pushed into ISTACK when entering an interrupt and popped out when exiting the interrupt.

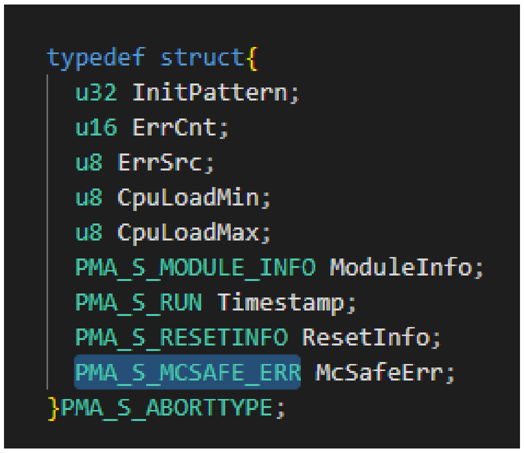
Main function



- The core0_main function is the main function for core 0, which will be executed after the execution of startup code
- In main function it first initializes Post Mortem Abort data. Then some startup checks are carried out by SMU, TsTM and TsThandler modules.
- The main function does the initialization of all the MCAL, BSW Modules and then finally enables the Interrupts. __asm ("enable").
- After which all the configured interrupts are serviced in a Nested Manner and the while loop never ends

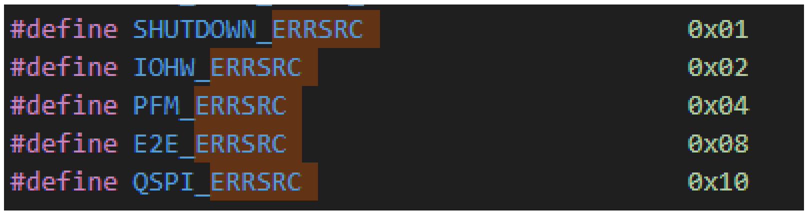
Postmortem Abort Data

- PMA Data is a structure of information about the error source, error count, cpuload, timestamp, reset info, Mcsafe error (microcontroller safety)



```
typedef struct{
  u32 InitPattern;
  u16 ErrCnt;
  u8 ErrSrc;
  u8 CpuLoadMin;
  u8 CpuLoadMax;
  PMA_S_MODULE_INFO ModuleInfo;
  PMA_S_RUN Timestamp;
  PMA_S_RESETINFO ResetInfo;
  PMA_S_MCSAFE_ERR McSafeErr;
}PMA_S_ABORTTYPE;
```

- It is used for analysing purposes if any issues are found
- PMA data is not stored in LVDC controller, it is sent to COM through internal CAN
- So for Eg: If any unintended reset happens, PMA stores the details about it (Eg: ErrSrc- Error Source :

  ```
  #define SHUTDOWN_ERRSRC                   0x01
  #define IOHW_ERRSRC                       0x02
  #define PFM_ERRSRC                        0x04
  #define E2E_ERRSRC                        0x08
  #define QSPI_ERRSRC                       0x10
  ```

  So by reading the value of ErrSrc in PMA structure one can identify the source of error, which can be used for debugging purposes)
- PMA init is called at the start of main function, here all the PMA data are reset to zero. In PMA initialization the status of last reset also should be checked (proper or improper reset). Reset status should be saved to ResetInfo.ShutdownSts in PMA structure
- For checking last reset was proper or improper, SCU_RSTSTAT register can be read, which gives the information about last reset. The value in this register is retained even after resets
  - PORST: Power On Reset
  - STM: System Timer Reset
  - EVR13: 1.3V
  - EVR33: 3.3V
  - SMU: Safety Management Unit can trigger alarms which can lead to reset
  - STBYR: Stand By Register, If device enters low power mode or sleep mode. LVDC controller enters sleep mode through COM if CAN is stopped. Current will go to zero, no communication.

Safety Management Unit

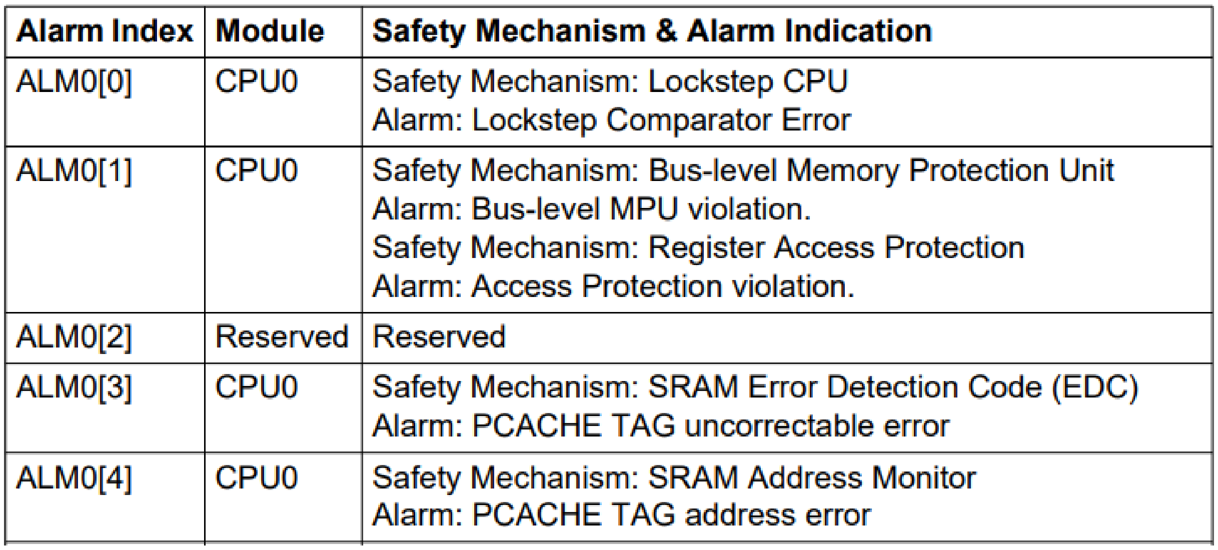
- SMU is a central component of the safety architecture providing a generic interface to manage the behaviour of the microcontroller under the presence of faults.
- The SMU centralizes all the alarm signals related to the different hardware and software-based safety mechanisms.
- Each alarm can be individually configured to trigger internal actions and/or notify externally the presence of faults via a fault signaling protocol.
- The alarm events are logged, the logged information is resilient to any system or application reset.

- Configuration of alarm is done using Tresos. In SMU there are 7 Alarm groups and each group have 32 alarms in it, ie alarm registers each of 32 bit.
- SMU checks is carried out before initialization in the main program
- Once an Error is detected appropriate reaction must take place like a Reset.

  SMU Module Consists Of :

  Config : To configure the Alarm and the Alarm Reactions (Done using tresos)

  Static : Code provided by Infineon
- Every alarm has a specific reason, some can be configured through the SW and the reaction for the alarm can be specified. For Each alarm configured we define a test provided by Infineon
- Eg for alarms:



Alarm Index	Module	Safety Mechanism & Alarm Indication
ALM0[0]	CPU0	Safety Mechanism: Lockstep CPU Alarm: Lockstep Comparator Error
ALM0[1]	CPU0	Safety Mechanism: Bus-level Memory Protection Unit Alarm: Bus-level MPU violation. Safety Mechanism: Register Access Protection Alarm: Access Protection violation.
ALM0[2]	Reserved	Reserved
ALM0[3]	CPU0	Safety Mechanism: SRAM Error Detection Code (EDC) Alarm: PCACHE TAG uncorrectable error
ALM0[4]	CPU0	Safety Mechanism: SRAM Address Monitor Alarm: PCACHE TAG address error

- After configuring the SMU (init) to check whether it is working properly or not TsTM and TsT handler will together perform fault injection tests to check whether SMU has taken the required action or not.
- IOHWSF_vMcSafeStartUpChecks();

  // Starts microcontroller safety tests, Inside this TsT is initilaized, and pre run tests are carried out. Thes tests are provided by Infineon, we do some manual modifications such as adding error handlers.
  - TsTM_Init: Initialization of the Test Manager global variables and invoke the initialization of SafeTlib. It calls SMU init, where the alarms which we are using are configured.
  - TstM_PreRunTst: Starts the tests. There are a set of predefined tests for checking the functioning of SMU. They are divided into different groups.
- SMU Module States
  - Init

    - Run
    - Post Run
- SMU runs tests on other peripherals in order to find they are working properly or not. The startup checks during init phase are done to ensure that the SMU is able to detect the errors. The tstm and tsthandler modules will inject faults and observe whether smu is able to detect these or not. If SMU is able to detect then proceed to run state assuming SMU will detect the errors. If the startup checks fails, the controller wont go to run state instead go to a safe state by performing a reset
- TstM (Test Manager): Contains function from Infineon Safety Library. SW Module is the caller of the test handler to execute the tests. It does application level checks and consolidates the results of program flow. It also invokes SMU for triggering the safe state and Error handling through Fault Injection Tests
- Test handler inject the failure and SMU we configure what reactions are required for different errors

PORTS

- Each port pin can be configured for input or output operation
- Switching between input and output mode is accomplished through the Pn_IOCR register, which enables or disables the output driver. The port will be in input mode after reset (default)
- Pn_IOCR- Input output control register, Sets in input/output mode, also selects pin muxing. Values for muxing can be found in datasheet
- Pn_OMSR- Set Pn_Out, Output Modification Set Register
- Pn_OMCR- Clear Pn_Out Output Modification Set Register
- Pn_OMR- Output Modification Register, Set Clear Toggle or remain unchanged
- Pn_In- Reading input
- Pn_Out- Port output
- Port_Cfg – All port configuration is done here, setting pin as input/output along with peripheral muxing
- Emergency stop: Feature provide by Infineon to stop entire port group in case of any failures and achieve safe state
-

GTM – Generic Timer Module

- TOM: Timer output module, TOM channel is able to generate a PWM signal at its output. TOM0 and TOM1 with each having 16 Channels to generate PWM

- TIM: Timer input module, Can be used to measure PWM Duty Cycle, Period of incoming PWM.
- GTM: TOM module is used as Interrupt Source for Scheduler and PWM